سید محمد طاها طباطبایی – تمرین سری ششم

۹۸۱۲۷۶۲۸۳۸

چکیده:

در پارت اول، به بررسی فضای رنگی HSI و تبدیل و مقایسه آن با RGB می‌پردازیم. در پارت بعدی، ۳ نمونه فضای رنگی جدید معرفی شده است.

در بخش دوم، اثر سطح‌بندی تصویر و استفاده از بیت های کمتر از ۸ در نمایش تصاویر خاکستری و رنگی بررسی شده است. در انتها نیز به بررسی تاثیر نمایش تصویر با تعداد حالت های رنگی متفاوت و وزن دهی متفاوت به سطح ها در کانال های رنگی متفاوت پرداخته شده است.

توضیحات فنی:

- توضیح جزییات پیاده‌سازی توابع در محل پیاده‌سازی کد، به صورت کامنت و داکیومنت نوشته شده است.

۶.۱.۱

در سلول اول، با استفاده از تابع RGB2HSI، تصویر را از فضای رنگی RGB به HIS منتقل می‌کند. برای پیدا کردن هر کدام از سه مشخصه، hue، saturation و intensity از توابعی استفاده شده است که طبق فرمول معرفی شده در اسلاید های درس، محاسبات را انجام می‌دهند. در گام اول، اگر از سه کانال رنگی RGB برای بررسی استفاده کنیم، متوجه می‌شویم میزان حضور رنگ قرمز در تصویر بیشتر است. حضور رنگ قرمز بیشتر از ۲ رنگ دیگر، به این معنی است که احتمالا در نمایش فام، باید پیکسل های بسیار سیاه یا سفید بیشتری داشته باشیم، چون در نمایش فام، زاویه های نزدیک به صفر یا ۳۵۹ درجه به رنگ قرمز اختصاص یافته است، و بر اساس اینکه در آن محل، رنگ قرمز بیشتر با سبز ترکیب شده یا آبی، زوایا به صفر یا ۳۵۹ نزدیکتر خواهند بود.

$$\theta=\cos-1\{\ 12[(R-G)+(R-B)][\ (R-G)2+(R-B)(G-B)\ ]1/2\ \} \qquad (7\text{-}17)$$

The saturation component is given by

$$S=1-3(R+G+B)[\min(R,G,B)] \qquad (7\text{-}18)$$

Finally, the intensity component is obtained from the equation

$$I=13(R+G+B) \qquad (7\text{-}19)$$

مشخصه I در فضای HIS، نشان دهنده بیشترین اطلاعات از تصویر است. در تصویر باند I، نقاط روشن‌تر، نقاطی هستند که شدت رنگ بیشتری داریم، و نقاط تیره‌تر، نقاطی است که مجموع مقادیر پیکسل های رنگی در آن نقاط کم است.

مشخصه S، اشباع را نشان می‌دهد. اشباع، میزان خلوص رنگ در فضای رنگی است. به طوری که هرچه مقدار حضور یک رنگ، بیشتر از مقدار حضور رنگ های دیگر باشد، اشباع بیشتری داریم. به طور مثال لبه کلاه، چون یک ناحیه است که از ترکیب تقریبا یکسانی از سه رنگ تشکیل شده است، اشباع کمی دارد، برای همین به رنگ تیره تر تبدیل شده.(figure 1)

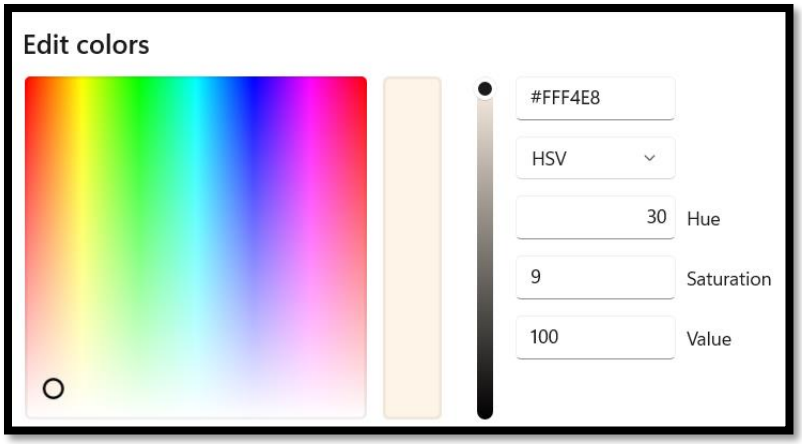


Figure 2 – فام و اشباع کم، درحالی که شدت روشنایی زیاد است.

برخی نقاط تیره، در شاخصه اشباع مقدار بسیار سفیدی پیدا کرده اند، زیرا علی رغم اینکه مقدار intensity زیادی ندارند و به ظاهر تیره هستند، اما چون اشباع رنگ آبی آن ها زیاد است، در شاخصه اشباع پیکسل های روشنی را به خود اختصاص داده اند.(figure 2)

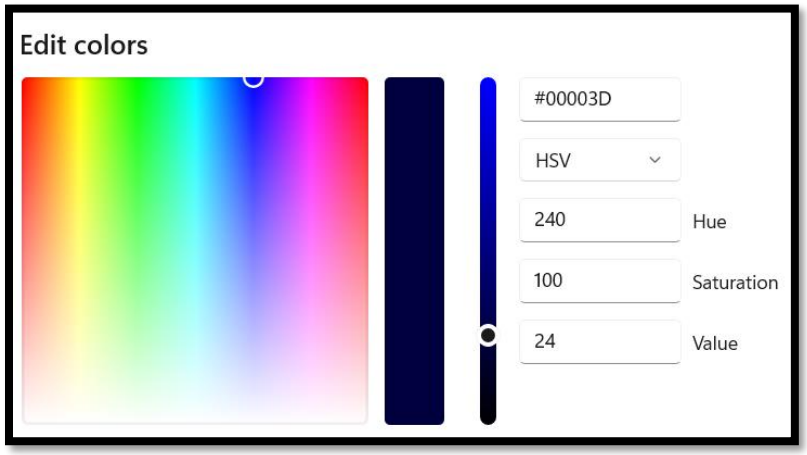


Figure 2 – اشباع ۱۰۰ درصدی با رنگ آبی تیره

مشخصه h، فام را مشخص می‌کند. نقاط تیره در نتیجه نمایش داده شده، نقاط قرمزی هستند که زوایای نزدیک به صفر دارند، و نقاط روشن، نقاط قرمز نزدیک به زاویه ۳۵۹. مقادیر دیگر نیز به همین ترتیب، از رنج ۰ تا ۳۵۹ به ۰ تا ۲۵۵ مپ می‌شوند. به طور مثال در قسمت پر روی کلاه که رنگ آبی دارد، رنگ خاکستری روشن داریم که تقریبا همان محدوده رنگ آبی است. قسمت پوست رنگ های تیره است، که به دلیل وجود رنگ قرمز و زرد است. اینکه در این تصویر رنگ های خاکستری میانی زیادی نداریم و شاید تصویر کمی باینری سفید و سیاه است، به این دلیل است که رنگ های میانی، نماینده رنگ سبز و فیروزه ای است که در این تصویر، این رنگ ها کمتر مشاهده می‌شود.

۶.۱.۲

HWB

مخفف رنگ(hue) ، سفیدی (whiteness) و سیاهی (blackness) است. مانند HSL، رنگ می‌تواند هرجایی در محدوده ۰ تا ۳۶۰ باشد. دو آرگومان دیگر، میزان رنگ سفید یا سیاه را تا ۱۰۰٪ کنترل می‌کنند (که منجر به یک رنگ کاملاً سفید یا کاملاً سیاه می‌شود). اگر مقادیر مساوی سفید و سیاه باهم میکس شوند، رنگ حاصل، خاکستری است. ما می‌توانیم این را شبیه به ترکیب کردن رنگ در نظر بگیریم، که برای ایجاد پالت‌های رنگی تک‌رنگ، مفید است.

Adobe RGB

رنگ‌ها و یا تون‌های رنگی مختلف بیشتری نسبت به RGB دارد. در مقایسه با RGB این فضای رنگی ۳۵٪ محدوده رنگی بیشتری دارد البته برخی منابع ادعا می‌کنند محدوده رنگ‌هایAdobe RGB شبیه به RGB (بیش از ۱۶ میلیون رنگ) است و تنها تفاوت این دو در تعداد رنگ‌هایی است که هرکدام از آن‌ها می‌توانند نمایش دهند. Adobe RGB فضای رنگی ایده‌آل برای عکاسی است.

LCH

مخففlightness ، chromaو hue است. مانندLAB ، درصد روشنایی می‌تواند تا ۱۰۰ باشد. مشابهHSL ، رنگ یا hue می‌تواند محدوده‌ای بین ۰ تا ۳۶۰ باشد .کروما میزان رنگ را نشان می‌دهد، و ما می‌توانیم آن را شبیه به اشباع در HSL در نظر بگیریم. اما chroma می‌تواند از ۱۰۰ تجاوز کند و درواقع، از نظر تئوری نامحدود است. استفاده از LCH به ما امکان دسترسی به طیف وسیع‌تری از رنگ‌ها را می‌دهد.

۶.۲.۱

برای پیاده‌سازی قطعه‌بندی از تابع quantize استفاده شده است. ورودی این تابع، تصویر و تعداد بیت های در دسترس برای سطح‌بندی است. این تابع، با تقسیم۲۵۶ بر 2^{n_bits} ، تعداد دسته های مورد نظر را محاسبه می‌کند. سپس پیکسل های تصویر را بر این مقدار تقسیم و به پایین گرد می‌کند. در نهایت حاصل را دوباره در coeff ضرب می‌کنیم. منطق این عملیات این است که ابتدا پیکسل ها به اعدادی بین ۰ تا level-1 مقداری که میخواهیم نرمالایز می‌شوند، و در ادامه دوباره در coeff ضرب می‌کنیم تا به اعداد ابتدای بازه ها مپ شوند. برای ۶۴ سطح، به ۶ بیت، ۳۲ سطح به ۵ بیت، ۱۶ سطح به ۴ بیت و ۸ سطح به ۳ بیت نیاز داریم. ۴ تصویر حاصل را نمایش دادیم. مشخص است که تفاوت چشم‌گیری بین تصویر اصلی و تصاویر با۶۴، ۳۲ یا ۱۶ سطح خاکستری وجود ندارد و چشم ما این تفاوت را تشخیص نمی‌دهد. فقط در تصویر آخر تغییر کیفیت تصویر قابل لمس است.

L = 64	*L = 32*	*L = 16*	*L =8*	
42.663301	35.803170	29.163439	23.169236	*PSNR*
3.521683	17.090836	78.838150	313.441666	*MSE*

نتایج مقایسه نشان می‌دهد که MSE در تصویر ۸ سطحی نسبت به بقیه تصاویر یک جهش در مقدار تفاضل دارد که با نتایج شهودی به دست آمده نیز منطبق است. کمترین مقدار PSNR نیز برای همین تصویر به‌دست آمد.

۶.۲.۲

برای سطح‌بندی تصاویر رنگی، باید تابع سطح‌بندی را روی هر کانال تصویر به طور مجزا اعمال کنی. کانال قرمز و سبز را در ۳ بیت و کانال آبی را در ۲ نشان دادیم. به طور مشخص، کیفیت سایه های کاهش محسوسی داشته. همچنین در بعضی قسمت ها رگه های رنگ آبی پدید آمده که به دلیلی این است که رنگ آبی در یک بیت کمتر نمایش داده می‌شود و گاهی نسبت به پیکسل های قرمز و سبز اطراف خود به سطح بالاتری گرد شده است، مانند رگه آبی روی بازو مدل.

۶.۲.۳

با توجه به اینکه می‌توان برای حالت های ۳۲ رنگی و ۱۶ رنگی، حالت های مختلفی متصور شد که در هر کدام، تعداد بیت متفاوتی به کانال های رنگی اختصاص داد، تمام این حالات را محاسبه کردیم. در حالت ۳۲ رنگی، ۳ تصویر خروجی، هرکدام نمایش دهنده حالتی هستند که یک کانال رنگی با ۱ بیت و دو کانال رنگی با ۲ بیت سطح‌بندی شده است. در هر کدام از تصاویر، کانالی که به آن یک بیت اختصاص یافته است، آن رنگ را ضعیف تر نمایش می‌دهد. به طور مثال در تصویری که کانال آبی ۱ بیتی است، تصویر به سمت رنگ زرد متمایل است. تصویری که ۱ بیت قرمز دارد، فیروزه ای رنگ و تصویری که ۱ بیت سبز دارد، بنفش تر به نظر می‌رسد.

مشابه حالت بالا، برای نمایش ۱۶ رنگ متفاوت نیز، ۳ تصویر با ۳ تقسیم بیت متفاوت برای کانال های رنگی داریم. در اینجا نیز، تصویری که ۲ بیت برای رنگ قرمز دارد، قرمزتر، تصویری که ۲ بیت برای سبز دارد، سبزتر و تصویر آخر نیز به دلیل اختصاص ۲ بیت برای رنگ آبی، آبی تر دیده می‌شود.

در حالت ۸ رنگه نیز، به هر کانال رنگی ۱ بیت اختصاص داده ایم.

```
import numpy as np
import math
import cv2
import matplotlib.pyplot as plt
from skimage.metrics import peak_signal_noise_ratio


def RGB2HSI(image):
    """
    Transform image from RGB space to HSI. Calculates Hue, Saturation &
Intensity.

    Inputs:
        - image: The source image.

    Returns:
        Image in HSI space.
    """

    # suspend errors: 1) divide by zero 2) inavalid data-type
    with np.errstate(divide='warn', invalid='warn'):

        # normalize pixels in range [0,1]
        image_normal = np.float32(image)/255.0

        # Separate color channels
        # typr:ignore
        red = image_normal[:,:,0]
        # type: ignore
        green = image_normal[:,:,1]
        # type: ignore
        blue = image_normal[:,:,2]

        h = hue(red.copy(),green.copy(),blue.copy())

        s = saturation(red.copy(),green.copy(),blue.copy())

        i = intensity(red.copy(),green.copy(),blue.copy())

        return h,s,i


def hue(red, green, blue):
```

```
    """
    Calculate the hue(H) in the HSI color space from the RGB channels.
Hue is calculated
    according to the following formula.

    H =
    θ
        if (B ≤ G)
        ,else:
            (360 - θ)


    where:

    θ = arccosine
        {
        1/2 * [(R - G) + (R - B)]
        /
        sqrt( power((R - G),2) + (R - B)*(G - B) )
        }

    Inputs:
        - red: Red channel in RGB space.
        - green: Green channel in RGB space.
        - blue: Blue channel in RGB space.

    Returns:
        The hue.

    """

    # numerator of equation
    numerator =
np.multiply(0.5,(np.add(np.subtract(red,green),np.subtract(red,blue))))

    # denominator of equation
    z1 = np.add(np.power(np.subtract(red,green),2),
np.multiply(np.subtract(red,green),np.subtract(green,blue)))
    z1 = np.where(((z1-0) <= 0.00001) , 0.00001 , z1)
    denominator = np.sqrt(z1)

    # final division
    x = np.divide(numerator,denominator)
```

```
    # tetha in range [0,2*pi]
    tetha = np.arccos(np.divide(np.multiply(x,(math.pi*2)), 360))

    # if B <= G  h = tetha , else h = 2*pi - tetha
    hue = np.where(blue<=green , tetha , (math.pi*2.0 - tetha))

    return hue


def saturation(red, green, blue):
    """
    Calculate the saturation in the HSI color space from the RGB channels.
Saturation is
    calculated according to the formula below.

    S = 1 - ( 3 * [min( R, G, B)] / (R + G + B) )

    Inputs:
        - red: Red channel in RGB space.
        - green: Green channel in RGB space.
        - blue: Blue channel in RGB space.

    Returns:
        The saturation.

    """
    minimum = np.minimum(np.minimum(red, green), blue)
    saturation =
np.subtract(1,np.multiply(np.divide(3,np.add(np.add(red,blue,green),0.0001
)),minimum))
    # saturation = 1 -
np.multiply(np.divide(3,np.add(red,blue,green)),minimum)

    return saturation


def intensity(red, green, blue):
    """
    Calculate the intensity in the HSI color space from the RGB channels.
Intensity is
    the average value of the three channels.
```

```
    Inputs:
        - red: Red channel in RGB space.
        - green: Green channel in RGB space.
        - blue: Blue channel in RGB space.

    Returns:
        The Intensity.
    """

    return np.divide((blue + green + red), 3)

def normalize(array,newMax,newMin):
    """
    A simple normalization function.

    Inputs:
        - array: Array to be normalized
        - newMax: Max of new range.
        - newMin: Min of new range.

    Returns:
        Normalized array.

    """
    if isinstance(array, list):
        return list(map(normalize, array,newMax,newMin))
    if isinstance(array, tuple):
        return tuple(normalize(list(array),newMax,newMin))
    normalizedData = (array-np.min(array))/(np.max(array)-
np.min(array))*(newMax-newMin) + newMin
    return normalizedData


def quantize(array, n_bits):
    """
    Image array(range of 0 to 255) is quantized to the desired number of
bits. Its actually generate
    a new array, assuming that the representation uses 'n_bits' instead of
the default 8 bits.

    Inputs:
        - array: The image matrix
        - n_bits: Given number of bits.
```

```
    Returns:
        The quantized array.
    """
    coeff = 2**8 // 2**n_bits
    return (array // coeff) * coeff


def mean_square_error(imageSource, imagetarget):
    """
    The "Mean Squared Error" between the two images is the
    sum of the squared difference between the two images.
    the lower the error, the more "similar" the two images are.

    NOTE: the two images must have the same dimension

    Inputs:
        - imageSource: The source image, we want to calculate the target
image difference of
        - imageTarget: The target image, we calculate how far it is from
the source

    Returns:
        The MSE
    """

    # cumulative difference
    err = np.sum((imageSource.astype("float") -
imagetarget.astype("float")) ** 2)

    # divide by length*width
    err /= float(imageSource.shape[0] * imageSource.shape[1])

    return format(err,'.6f')

def PSNR(srcImage,testImage):
    """
    Implementation of 'Peak Signal to Noise Ratio' method
    using, sci-kit image library.
    The greater the result, the more "similar" the two images are.

    Inputs:
```

```
        - srcImage: The source image, we want to calculate the target
image difference of
        - testImage: The target image, we calculate how similar it is with
the source

    Returns:
        The PSNR
    """
    return peak_signal_noise_ratio(srcImage,testImage)
```

In [ ]:
```
from functions import *
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
plt.style.use(plt.style.available[5])
```

6.1. Color space

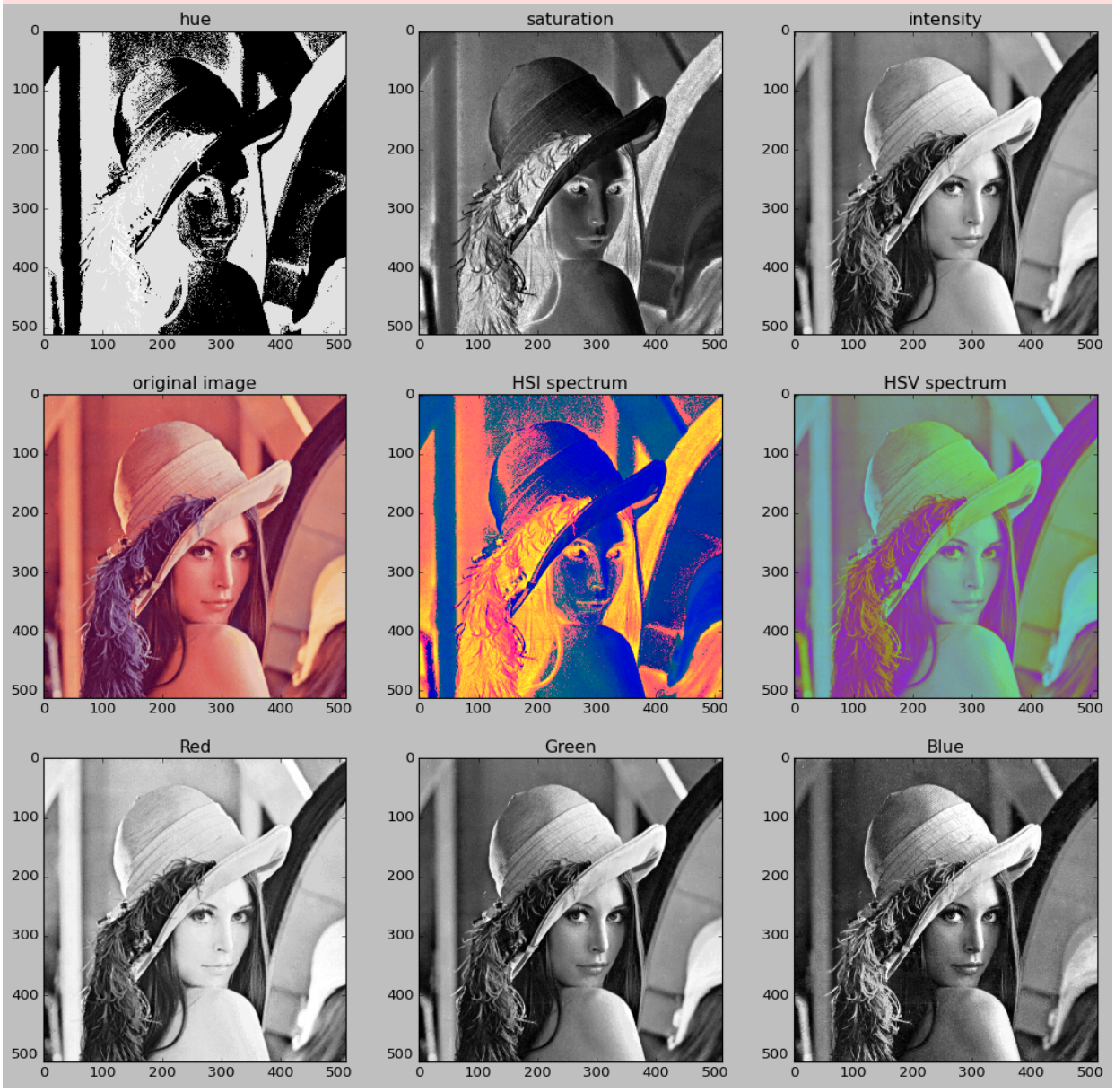
6.1.1. Convert Lena to HSI format, and display the HIS components as separate grayscale images. Observe these images to comment on what does each of the H, S, I components represent. The HSI images should be saved in double precision.

In [ ]:
```
lena = cv2.imread('Lena.bmp')
lena = cv2.cvtColor(lena, cv2.COLOR_BGR2RGB)

# red,green,blue = cv2.split(lena)
# lena_abs = cv2.merge((red,green,blue))

lena_hsv = cv2.cvtColor(lena,cv2.COLOR_BGR2HLS)


h,s,i = RGB2HSI(lena)
h = normalize(h,(math.pi*2),0)
s = normalize(s,1,0)
# i = normalize(i,255,0)
lena_hsi = cv2.merge((h,s,i))
```

In [ ]:
```
figure = plt.figure(figsize=(16,15))

figure.add_subplot(3,3,1)
plt.imshow(h,cmap='gray')
plt.title('hue',color='black')

figure.add_subplot(3,3,2)
plt.imshow(s,cmap='gray')
plt.title('saturation',color='black')

figure.add_subplot(3,3,3)
plt.imshow(i,cmap='gray')
plt.title('intensity',color='black')

figure.add_subplot(3,3,4)
ha,la,sa = cv2.split(lena)
plt.imshow(lena)
plt.title('original image',color='black')

figure.add_subplot(3,3,5)
plt.imshow(lena_hsi)
plt.title('HSI spectrum',color='black')

figure.add_subplot(3,3,6)
plt.imshow(lena_hsv)
plt.title('HSV spectrum',color='black')
```

```
red,green,blue = cv2.split(lena)

figure.add_subplot(3,3,7)
plt.imshow(red,cmap='gray')
plt.title('Red',color='black')

figure.add_subplot(3,3,8)
plt.imshow(green,cmap='gray')
plt.title('Green',color='black')

figure.add_subplot(3,3,9)
plt.imshow(blue,cmap='gray')
plt.title('Blue',color='black')

plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



6.1.2. *Present and discuss new color space (at least three) in detail which was not introduced in class (Application, Equation, etc.).

6.2. Quantization

In [ ]:
```
from functions import *
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
plt.style.use(plt.style.available[5])
```

6.2.1. Implement uniform quantization of a color image. Your program should do the following:

1. Read a grayscale image into an array.
2. Quantize and save the quantized image in a different array.
3. Compute the MSE and PSNR between the original and quantized images.
4. Display and print the quantized image.

Notice, your program should assume the input values are in the range of (0,256), but allow you to vary the reconstruction level. Record the MSE and PSNR obtained with L = 64, 32, 16, 8 and display the quantized images with corresponding L values. Comment on the image quality as you vary L. (Test on Lena Image).

In [ ]:
```
lena = cv2.imread('Lena.bmp',cv2.IMREAD_GRAYSCALE)

lena64 = quantize(lena.copy(),6)
lena32 = quantize(lena.copy(),5)
lena16 = quantize(lena.copy(),4)
lena8 = quantize(lena.copy(),3)
```

In [ ]:
```
figure = plt.figure(figsize=(11,10))

figure.add_subplot(2,2,1)
plt.imshow(lena64,cmap='gray')
plt.title('lena64',color='black')

figure.add_subplot(2,2,2)
plt.imshow(lena32,cmap='gray')
plt.title('lena32',color='black')

figure.add_subplot(2,2,3)
plt.imshow(lena16,cmap='gray')
plt.title('lena16',color='black')

figure.add_subplot(2,2,4)
plt.imshow(lena8,cmap='gray')
plt.title('lena8',color='black')
```

Out[ ]:
```
Text(0.5, 1.0, 'lena8')
```

In [ ]:
```
print(f'L=64: PSNR= {PSNR(srcImage=lena,testImage=lena64)}, MSE= {mean_square_error(im
print(f'L=32: PSNR= {PSNR(srcImage=lena,testImage=lena32)}, MSE= {mean_square_error(im
print(f'L=16: PSNR= {PSNR(srcImage=lena,testImage=lena16)}, MSE= {mean_square_error(im
print(f'L=8: PSNR= {PSNR(srcImage=lena,testImage=lena8)}, MSE= {mean_square_error(imag
```

```
L=64: PSNR= 42.66330132120192, MSE= 3.521683
L=32: PSNR= 35.80317064962315, MSE= 17.090836
L=16: PSNR= 29.16343936072466, MSE= 78.838150
L=8: PSNR= 23.169236343419712, MSE= 313.441666
```

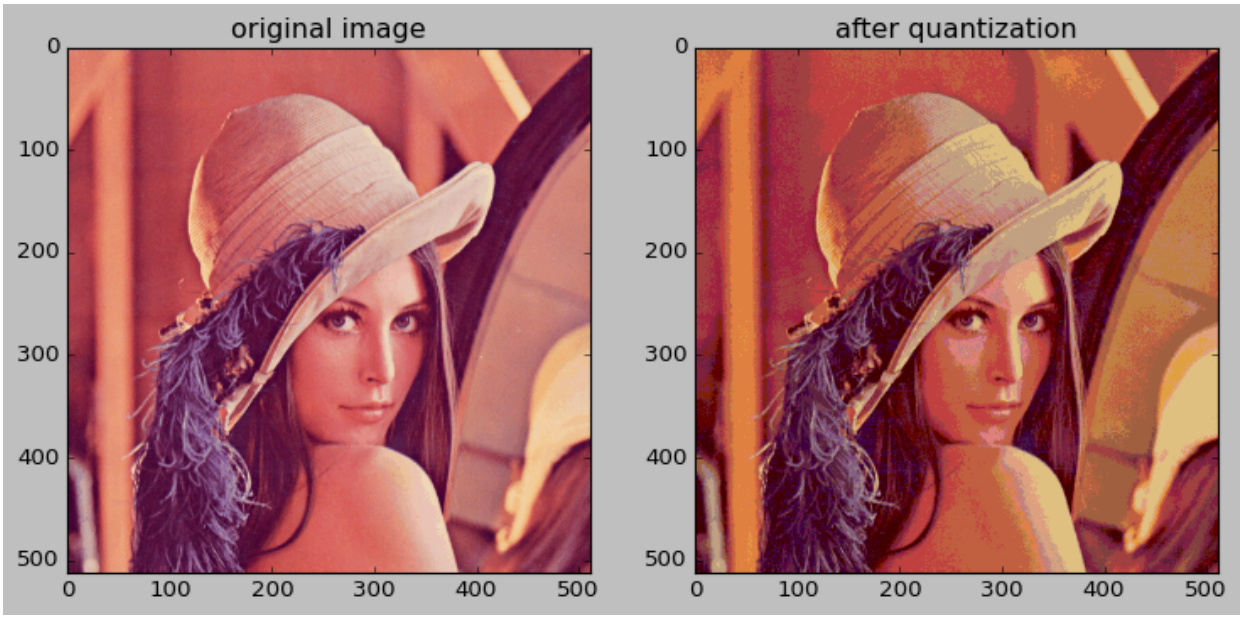
6.2.2. For the Lena image, quantize the R, G, and B components to 3, 3, and 2 bits, respectively, using a uniform quantizer. Display the original and quantized color image. Comment on the difference in color accuracy.

In [ ]:
```
lena = cv2.imread('Lena.bmp')
lena = cv2.cvtColor(lena, cv2.COLOR_BGR2RGB)
red,green,blue = cv2.split(lena)

newRed = quantize(red,3)
newGreen = quantize(green,3)
newBlue = quantize(blue,2)
```

```
lena_quantized = cv2.merge((newRed,newGreen,newBlue))
```

In [ ]:
```
figure = plt.figure(figsize=(11,10))

figure.add_subplot(1,2,1)
plt.imshow(lena)
plt.title('original image',color='black')

figure.add_subplot(1,2,2)
plt.imshow(lena_quantized)
plt.title('after quantization',color='black')
```

Out[ ]:
```
Text(0.5, 1.0, 'after quantization')
```

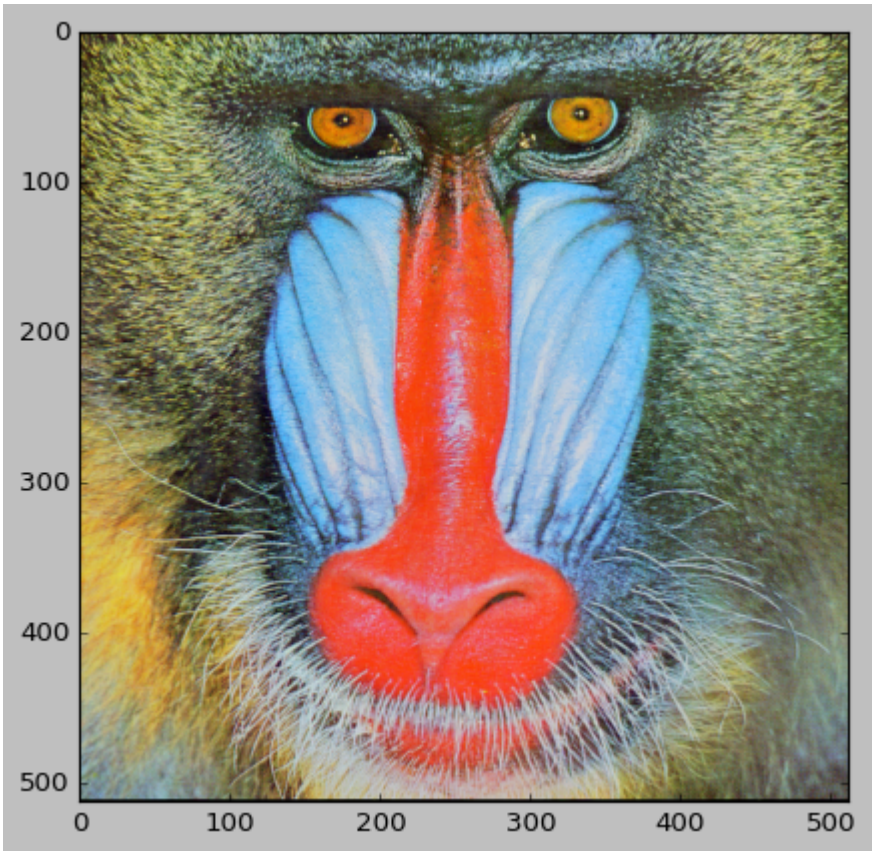


6.2.3. We want to weave the Baboon image on a rug. To do so, we need to reduce the number of colors in the image with minimal visual quality loss. If we can have 32, 16 and 8 different colors in the weaving process, reduce the color of the image to these three special modes. Discuss and display the results.

Note: you can use immse and psnr for problem 6.2.

In [ ]:
```
baboon = cv2.imread('baboon.bmp')
baboon = cv2.cvtColor(baboon, cv2.COLOR_BGR2RGB)
red,green,blue = cv2.split(baboon)

plt.imshow(baboon)
```

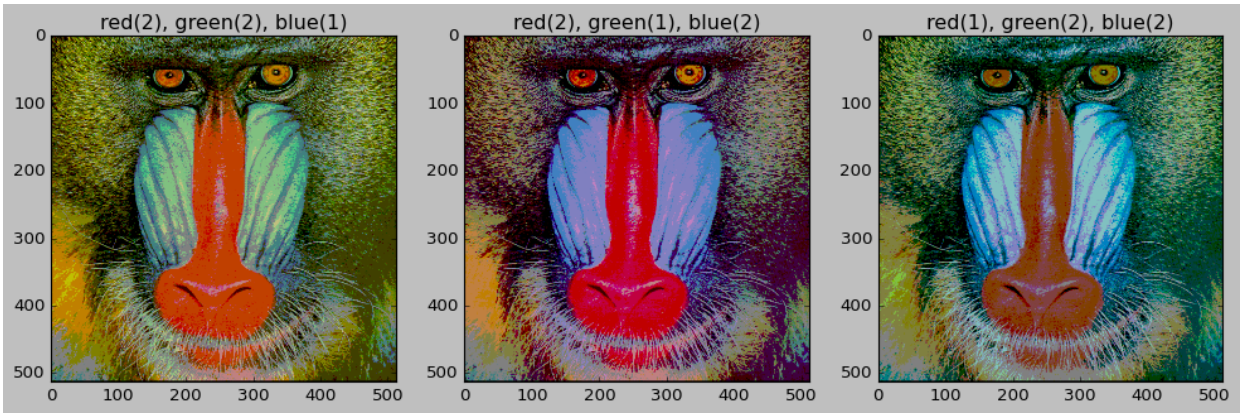
Out[ ]:
```
<matplotlib.image.AxesImage at 0x1ca9df61270>
```



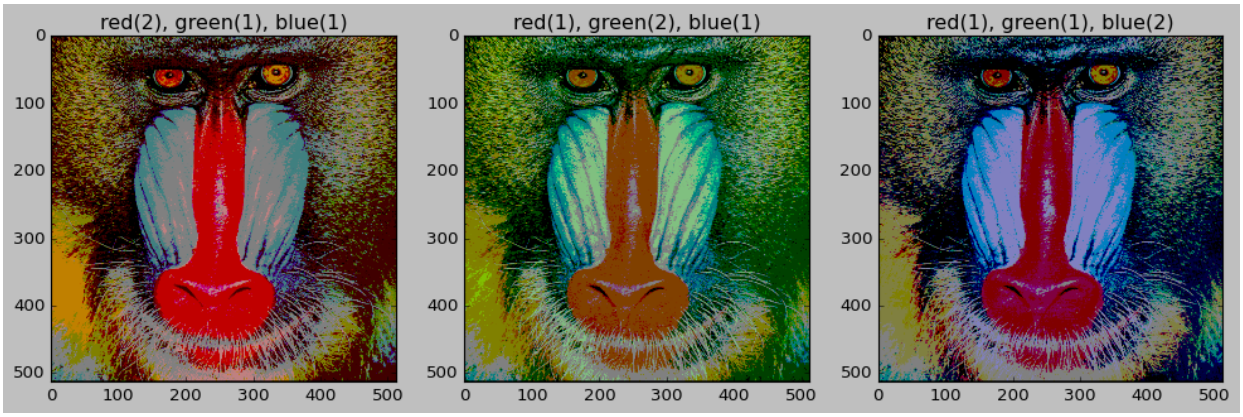
In [ ]:

```
figure = plt.figure(figsize=(15,15))

# for 32 colors
newRed = quantize(red,2)
newGreen = quantize(green,2)
newBlue = quantize(blue,1)

baboon32colors = cv2.merge((newRed,newGreen,newBlue))

figure.add_subplot(1,3,1)
plt.imshow(baboon32colors)
plt.title('red(2), green(2), blue(1)',color='black')


newRed = quantize(red,2)
newGreen = quantize(green,1)
newBlue = quantize(blue,2)

baboon32colors = cv2.merge((newRed,newGreen,newBlue))

figure.add_subplot(1,3,2)
plt.imshow(baboon32colors)
plt.title('red(2), green(1), blue(2)',color='black')

newRed = quantize(red,1)
newGreen = quantize(green,2)
newBlue = quantize(blue,2)

baboon32colors = cv2.merge((newRed,newGreen,newBlue))

figure.add_subplot(1,3,3)
plt.imshow(baboon32colors)
plt.title('red(1), green(2), blue(2)',color='black')
```

```
plt.show()
```



In [ ]:
```
figure = plt.figure(figsize=(15,15))

# for 16 colors
newRed = quantize(red,2)
newGreen = quantize(green,1)
newBlue = quantize(blue,1)

baboon16colors = cv2.merge((newRed,newGreen,newBlue))

figure.add_subplot(1,3,1)
plt.imshow(baboon16colors)
plt.title('red(2), green(1), blue(1)',color='black')


newRed = quantize(red,1)
newGreen = quantize(green,2)
newBlue = quantize(blue,1)

baboon16colors = cv2.merge((newRed,newGreen,newBlue))

figure.add_subplot(1,3,2)
plt.imshow(baboon16colors)
plt.title('red(1), green(2), blue(1)',color='black')

newRed = quantize(red,1)
newGreen = quantize(green,1)
newBlue = quantize(blue,2)

baboon16colors = cv2.merge((newRed,newGreen,newBlue))

figure.add_subplot(1,3,3)
plt.imshow(baboon16colors)
plt.title('red(1), green(1), blue(2)',color='black')

plt.show()
```



In [ ]:
```
# for 8 colors
newRed = quantize(red,1)
newGreen = quantize(green,1)
newBlue = quantize(blue,1)

baboon8colors = cv2.merge((newRed,newGreen,newBlue))

plt.imshow(baboon8colors)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1ca9e3bd6f0>
```